DEVELOPMENT OF A TASK-LEVEL ROBOT PROGRAMMING AND SIMULATION SYSTEM*

H. Liu, K. Kawamura, S. Narayanan**, G. Zhang, H. Franke & M. Ozkan
Center for Intelligent Systems, Vanderbilt University
Nashville, Tennessee 37235, USA

H. Arima
Process Control Equipments Design Dept.
Tokico, Ltd., Kawasaki, JAPAN

ABSTRACT

This paper presents an ongoing project in developing a Task-Level Robot Programming and Simulation System (TARPS). The objective of this research is to design a generic TARPS that can be used for a variety of applications. Many robotic applications require off-line programming, and a TARPS is very useful in such cases. Task-level programming is object-centered in which user specifies tasks to be performed instead of robot paths; graphics simulation provides greater flexibility and also avoids costly machine setup and possible damage. A TARPS has three major modules: world model, task planner and task simulator. The system architecture, design issues and some preliminary results are given in this paper.

I. INTRODUCTION

Robot programming systems can be divided into three broad categories: guiding systems in which the user leads a robot through the motion to be performed, robot-level programming systems in which the user writes a computer program specifying motion and sensing, and task-level programming systems in which the user specifies operations by their desired effects on objects [8]. Guiding systems are primitive, e.g., there are no loops, conditionals, or sensors, but are easy to use and can be implemented without a general-purpose computer. Robot-level programming systems have the capabilities lacked by guiding systems; however, the user must be familiar with both computer programming and robot manipulation. Task-level programming is an attempt to shift the burden of detailed robot programming from the user to the computer where only goals or tasks need to be specified by the user. Recently, more and more robotic applications require robot programming to be done off-line. This is often complicated by frequent task change and critical timing requirement. A Task-Level Robot Programming and Simulation System (TARPS) can be very useful in such cases since task-level progrmming is much more efficient than robot-level programming and guiding, and through computer simulation extensive experiments and analyses can be performed without the high cost of machine setup and risk of damage.

The system architecture of the TARPS being developed consists of three major modules: world model, task planner and task simulator as shown in Fig. 1. Based on the world model, the task planner translates object-centered task specifications to appropriate robot motion sequences. The robot motion se-

* This work was supported in part by Tokico, Ltd.

** Mr. Narayanan is now with Dept. of Elec. & Comp. Engr., Univ. of Calif., Davis, CA 95616.

PRECEDING PAGE BLANK NOT FILMED

quences and the world model can be simulated on the graphics terminal. The high-level task planning part is implemented in LISP, object model in FLAVORS, and robot motion synthesis in FORTRAN. Certain important issues such as object representation, collision avoidance and trajectory planning are also addressed.

II. WORLD MODEL

Task-level programming and simulation cannot be achieved without a world model. This world model should include a robot model, physical object model and environment model. The robot is modeled as a mechanical linkage system with various joint parameters and constraints. These parameters and spatial geometry of the manipulator are needed to compute and simulate robot motion. 3-D object representation has been a major research topic in computer vision, CAD/CAM and computer graphics. A 3-D object can be represented by one of the three general classes: (1) surface or boundary, (2) sweep and (3) volume. We adopted a surface-based representation scheme based on the evaluation in [4] and the following reasons:

1) We plan to derive the object model from the CAD model, and the sweep representation has been used only to a very limited extent in CAD.

2) Surfaces can be recognized by vision or range sensors and so any representation scheme utilizing surface descriptions can be easily integrated into a sensor-based robot planning system.

Any 3-D planar object can be represented by a graph where every vertex, edge and surface of the object corresponds to a node in the graph, and the arcs of the graph are the connected_to relations. Each node can be implemented as a computational object with certain slots. For example, a vertex node may have slots for vertex_id, x, y, and z coordinates, and related_edges. Data from other nodes can be obtained by message sending. At present, the object model is entered through a menu-driven interface; simple object models such as rectangular blocks, cylinder, cone and sphere can be entered through the menu. Once simple object models are defined, composite objects can be constructed in a way similar to that of Constructive Solid Geometry (CSG) representation. Algorithms for constructing object model from CAD data and B-splines representation are under development. The environment model has a world frame and coordinate frames for objects and robots. Semantic network and homogeneous transform are used to express their positions with respect to one another.

A. Surface-Based Object Representation

The surface representation of objects makes use of "faces." Any "face" of the object can be considered as a subset of the enclosing surface or boundary of the object. Conversely, the union of all possible "faces" of an object constitutes the boundary of that object [11]. For 3-D planar object the representation primitives are boundary, faces, edges and vertices. The primitive of a "physical object" is represented as a "computational object" [12]. A computational object is typically characterized by a set of "instance-variables" whose values are used to determine the current state of the object and its relation with other objects. Furthermore, there are procedures which can be used to determine the attributes of other objects and to make decisions to schedule operations concerning that object. Listed below are some of the

information required to characterize the face. Each face of the object is represented using the same scheme as is used for other primitives.

face_id: Used for the purposes of identification of a face.

adjacent_to: Has a list of face_id's as its value.

normal_vector: Equation of the vector normal to the face. This value can be computed from the face vertex positions of a planar surface. For a quadric surface, however, we require information about the surface shape.

related_edges: A list of edge_id's that belong to the face.

B. Conversion from CAD Data

The modelling scheme described earlier requires a lot of information to represent the object. There is an increasing need to obtain such information from a CAD data base of the task environment [4]. For example, the dimensions of a rectangular block can be obtained from a CAD model of the block. From these dimensions, it is possible to select a reference frame for the block and then compute the relevant information such as: 1) Location of vertices with respect to the assigned object reference frame; 2) labelling of edges by applying a rule that any two vertices constitute an edge; 3) labelling of faces by computing those sequences of edges which form loops, i.e., staring from any vertex, find those edges which, when traversed, lead us back to the initial vertex without going through any vertex twice; and 4) computation of the face normal vector from the equation formed by the plane described by the face's vertices.

III. TASK PLANNING

The role of the task planner is to take task-level specifications from the user and generate manipulator-level specifications which can be used for simulation or sent to the robot controller. Task specifications may appear in various forms ranging from a pair of initial and goal states to an explicit sequence of subtasks. If only a goal is given, the task planner would need substantial domain knowledge in order to generate sequence of subtasks. An intermediate step might be to let task planner check task specifications and provide recommendations. Task planning can be carried out in two steps: task synthesis and robot motion synthesis. At the top-level is a task manager responsible for the selection and coordination of task skeletons, procedures to perform specific subtasks. It is also plausible to use a robot-level programming language between task synthesis and robot motion synthesis, i.e., task specification (high level) --> robot program (medium level) --> robot motion (low level). The robot-level programming should be accessible by the user.

Robot motion synthesis depends largely on type of applications. For example, assembly operations require compliant and guarded motion, while arc welding and spray painting operations require accurate motion and trajectory control. Object and world model is indispensable in robot motion synthesis. We attach homogeneous transform to symbolic spatial relation to express quantitative relation among objects and robot. The object and world model changes

dynamically as task goes on. An efficient and elegant way to update the model is by message sending in object-oriented paradigm. Collision avoidance is another important problem in robot motion planning which will be discussed in later sections.

We use a typical spray-painting robot to illustrate task synthesis and robot motion synthesis. Assume that five faces of a large rectangular block are to be painted except the face attached to a fixture which rotates the block.

A. User Interface

The user interface elicits information that is required from the user in order to accomplish the task. Typically, TARPS requires to know about the environment and objects. The user supplies data to completely define the object size, shape and location through the object representation primitives. This helps TARPS to configure the object and environment model. Besides information required to define the environment, the user also supplies some parameters that are required for the performance, monitoring and analysis of the task.

B. Task Synthesis

The task synthesizer acts as a scheduler and utilizes the relevant plan information for the performance of the task. The input to the task synthesizer comprises user-specified parameters defining the environment and the task. A given task, e.g., paint the block, is decomposed into a sequence of subtasks using heuristic planning rules, which are primarily concerned with the selection of task parameters to obtain a satisfactory task performance. One such heuristic rule is that "adjacent faces of an object are painted in sequence". The selection of adjacent faces, however, further depends on the constraints on the mobility of the work-piece and the reachability or work-envelope of the robot. After decomposition, each subtask is considered independently and manipulator paths are generated for each subtask in accordance with subtask constraints. An example of such a plan decomposition sequence for painting the block is: (1) paint face_1, (2) rotate block (+90), (3) paint face_2, (4) rotate block (+90), (5) paint face_3, (6) rotate block (+90), (7) paint face_4, (8) paint face_5.

C. PATH GENERATION

Generally speaking, for painting robot there are two types of motion: free motion and paint motion. Free motion comprises those motions between home position and initial positions of subtasks. Collision avoidance is usually the only concern in free motion. Paint motion, on the other hand, requires more accurate trajectory control. For example, the spray gun must always be perpendicular to the surface to be painted, and the distance between spray gun and the surface must remain constant. Once task parameters such as initial configuration, etc. have been determined, the task skeletons are responsible for calculating the path of the manipulator that can satisfactorily accomplish the task.

1. Motion Planning

Once the manipulator is at the initial position it is ready to start performance of the task. The path followed by the end-effector depends on the "paint-patterns." A paint pattern is the path that a spray gun attached to the end-effecter moves to deposit paint on the work-piece. Such a pattern depends on various parameters like shape of the work-piece, material properties of the paint, etc. Until now, generation of a painting pattern has been mostly experimental by using a guiding system. Such a trial and error approach is often time consuming and inefficient. We felt it necessary to develop an algorithm for the painting process. Algorithms for painting planar surface have been developed; those for painting curved surfaces can use polyhedral approximation and are under development.

Once a particular face of the work-piece has been painted, there is relative motion between the robot and the work-piece in order to position the manipulator at a suitable initial position to paint the next face as scheduled by the task planner. Fig. 2 shows the robot path from home position to the initial position for painting face_1. The inter-subtask motion has to take into account the following two factors: 1) Avoidance of collision during the manipulator and/or work-piece motion and 2) movement of the work-piece between subtasks to provide an easy access for the manipulator to paint the relevant face subject to the constraints on the freedom of work-piece to move in the workplace. Inverse kinemetics and collision detection are computed in this phase.

2. Collision Avoidance

There have been many studies relevant to the planning of a collision-free path. Two approaches have been used most often. One approach is "hypothesize-and-test" method which focused on algorithms for detecting collision among solids. Another approach consists of explicitly representing the set of those robot configurations which are collision-free [6],[7]. However, an efficient algorithm for computing a collision-free path for general robots is still not available.

We employ two strategies to plan collision-free paths in TARPS. The first one is using an heuristic method to plan a collision-free path which is similar to the second approach mentioned above. The other uses a collision detector to detect a possible collision during simulation. Whether the collision detector need to be executed can be determined by the high level task manager or by the user. Since two different motions are used here, we developed different algorithms for free and paint motion. In free motion we need to make sure that the end effector is in the safe space. In paint motion we consider other links since end effector is always some distance away from the workpiece. In inverse kinemetics computation, usually multiple solutions are obtainable. In such case we can select a collision-free solution. If none of them is collision free, then a different path must be generated.

IV. SIMULATION AND GRAPHICS

To evaluate and ensure good performance of the robot task planning we need to have the capability of analyzing the kinematics and dynamics of the robot manipulator. This also calls for the capability of presenting actual

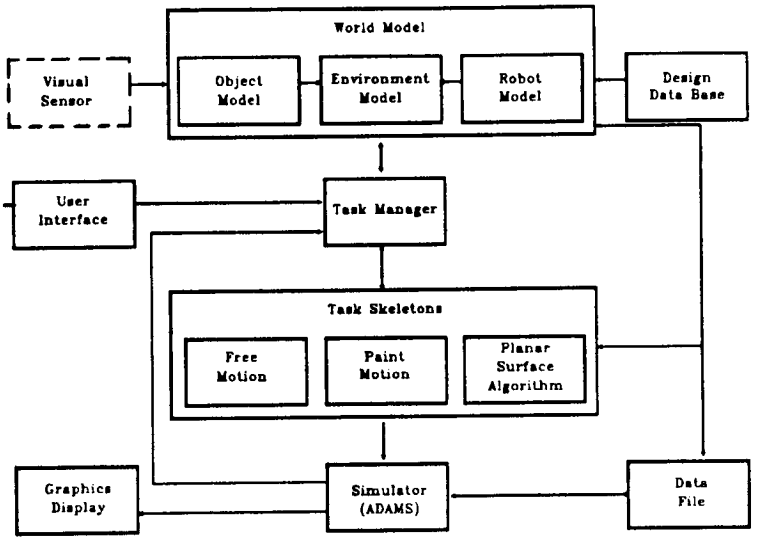

Figure 1: TARPS system architecture

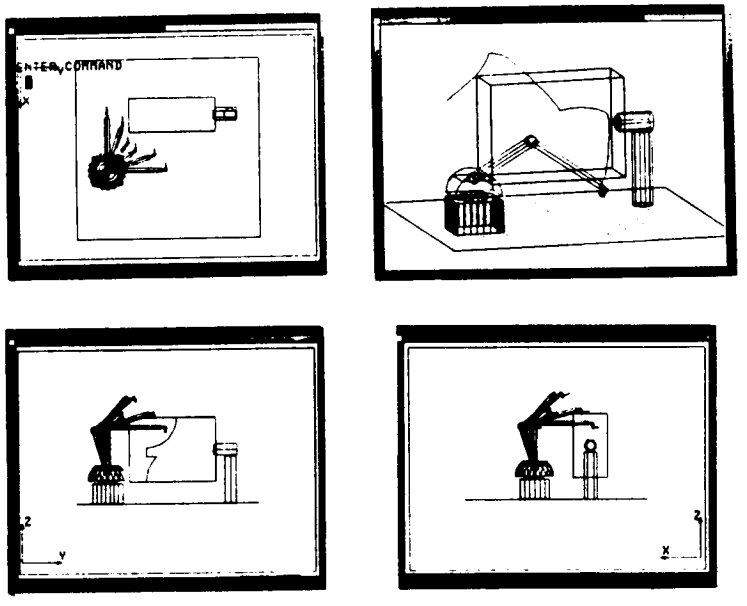

Figure 2: The complete (collision-free) motion sequence from x-direction to y-direction.

robot and workpiece motion graphically. The software used for this purpose must be extensible to encompass special requirements, e.g., nonlinear force and torque calculations, in the form of user-defined functions or subroutines. The Automatic Dynamic Analyses of Mechanical Systems (ADAMS) was selected as our simulation tool which provides us with such facilities. The actual equations governing the manipulator kinematics and dynamics can be programmed in ADAMS which can then be interfaced and run as a process when required by the LISP-based planner. The results of the simulation can be analyzed by the user or planning program to decide on the actions to take.

V. CONCLUSION

In this paper we have presented a prototype of Task-Level Robot Programming and Simulation System. The key issues addressed here are world modeling, object representation and collision-free task planning. We adopted a surface-based object representation for the reasons that it is ideal for sensor-based robot control and is easily accessible from CAD database. Task planning is based on a hierarchical approach while collision problem is taken into consideration during path synthesis. When sensor systems are incorporated, parts localization techniques can be applied to obtain actual position and orientation of the objects. It is our belief that task-level programming will be useful in a wide variety of applications, and we are also investigating the parallel hardware architecture for task-level systems.

VII. REFERENCES

1. ADAMS User's Manual, Mechanical Dynamics, Inc., 1985.
2. D.A. Ballard & C.M. Brown, Computer Vision, Prentice-Hall, Inc., New Jersey, 1982.
3. P.J. Besl & R.C. Jain, "Three-Dimensional Object Recognition," ACM Computing Surveys, 17(1):75-145, March, 1985
4. B. Bhanu & C-C. Ho, "CAD-Based 3D Object Representation for Robot Vision," Computer, 20(8): 19-35, August 1987.
5. M. Brady, J.M. Hollerbach, T.L. Johnson, T. Lozano-Perez & M.T. Mason, (Eds.), Robot Motion: Planning and Control, The MIT Press, Cambridge, Mass., 1983.
6. R.A. Brooks, "Solving the Find-Path Problem by Good Representation of Free Space," Proc. 2nd AAAI Conf., Carnegie-Mellon, August 1982.
7. T. Lozano-Perez, "Automatic Planning of Manipulator Transfer Movements," IEEE Trans. on System, Man, Cybernetics, SMC-11, 10, 1981.
8. T. Lozano-Perez, "Robot Programming," Proc. IEEE, 71(7): 821-841, July 1983.
9. R.P. Paul, "Manipulator Cartesian Path Control," IEEE Trans. on System, Man, and Cybernetics, SMC-9, 11, 1979.
10. R.P. Paul, Robot Manipulators, Mathematics, Programming, and Control, MIT Press, 1981.
11. A.A.G. Requicha, "Representations for Rigid Solids: Theory, Methods, and Systems," ACM Computing Surveys, 12(4):437-467, Dec. 1980.
12. D. Weinreb & D. Moon, LISP Machine Manual, Symbolics, Inc., 1981.